逢 甲 大 學

資訊工程學系專題報告

防盜電路

學　　生：　李 嘉 裕 (四丁)

指導教授：　徐 弘 洋

中華民國九十三年四月

目 錄

圖表目錄

第一章 導論

1.1 動機：

現代科技日益進步，許多家電都是幾乎使用單晶片製作，讓人想看看單晶片的功能及用處，加上在大三的時候，剛好學到單晶片8051 的相關原理和實作，於是感到很有興趣，加上大部份所學都是有關於計算機理論與軟體工程架構和技巧方面的知識，反而對硬體方面的認知與知識稍嫌不足，所以藉由此專題來對8051 有更深入的了解，並且對單晶片微電腦系統的功能與特性透過電路實作能有更深一層的認識，想看看一個小小的單晶片，如何可以運用在那些家電上。其中以MCS-51 此家族的功能更為強大以及便利使用，便以此為專題研討的方向。

1.2 目的：

1.對8051 單晶片硬體架構的了解。

2.對8051 單晶片所提供的指令集徹底了解。

3.文字型LCD 的架構與指令的研究。

4.如何使用8051 控制文字型LCD 。

5.利用8051 建構一個系統。

第二章 微電腦簡介

2.1 微電腦的基本架構

微電腦是由中央處理單元(CPU)，記憶體單元(memory)，及輸入輸出(Input/Output)三大部分所組成。而CPU 又分成兩大部分，分別是算數邏輯單元(ALU)以及控制單元(CU)。

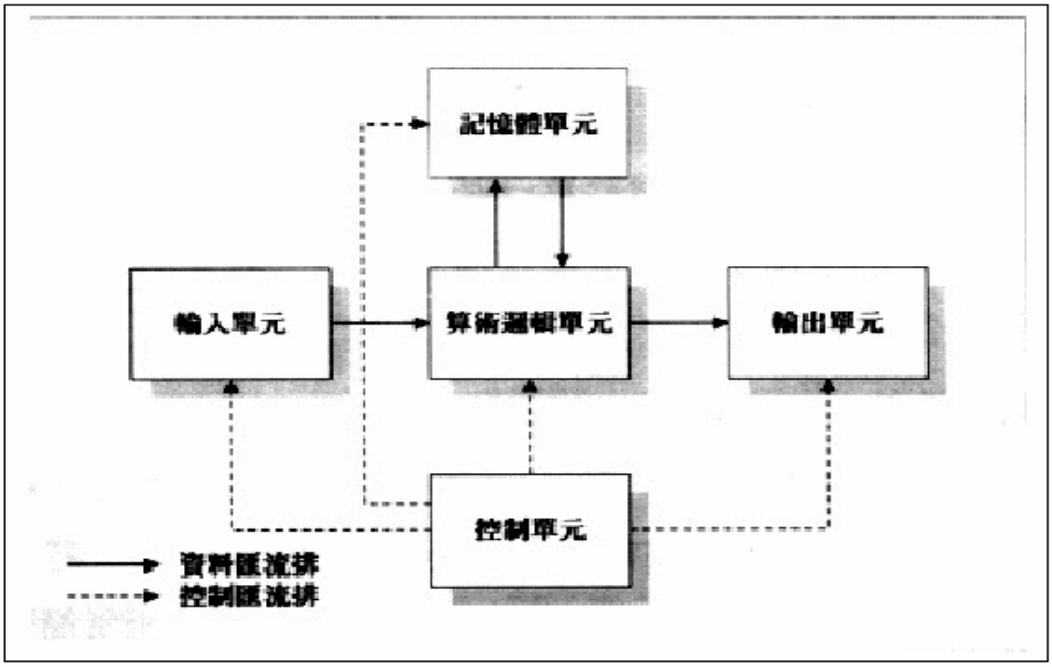


圖 2-1 微電腦基本架構

1.輸入單元(Input unit)：

負責將各種形式的資料或程式輸入到內部CPU 及 memory 中。

2.記憶單元(Memory unit)：

負責儲存程式資料或運算的結果，分成RAM(隨機存取記憶體)跟ROM(唯讀記憶體)。

3.算數邏輯單元(Arithmetic logic unit)：

負責計算比較和判斷等運算。

4.控制單元(Control unit)：

由記憶體中提取指令，加以解碼，產生控制信號。

5.輸出單元(Output unit)：

將電腦執行結果傳送到外界的裝置上。

2.2 單晶片微電腦

前面介紹的電腦系統，是由CPU 、記憶體以及I/O 所組成的，傳統的CPU 例如Z80 控制系統必須由許多控制晶片所組成：

1.CPU Z80

2.記憶體EPROM2764 容量8K 位元組。

3.記憶體RAM6264 容量8K 位元組。

4.I/O 控制晶片8255 。

5.計時計數控制晶片8253 。

6.串列傳輸控制晶片8251 。

7.中斷控制晶片8259 。

我們稱此為多晶片微電腦控制系統，主要用來設計較複雜的控制系統如影像處理、通訊控制及算術運算，由於使用晶片數量多，相對的硬體成本也提高。對於一些較簡易的控制系統，則不需要這麼多晶片來組成一個系統，於是出現了單晶片微電腦。單晶片微電腦是把五大基本單元，輸入單元，輸出單元，控制單元，記憶體單元以及算數邏輯單元濃縮在單一顆晶片上面，只要加上少許的電子零件便可以組成簡易的控制系統。單晶片微電腦上的ROM 以及RAM 的容量較小，因此其主要功用是應用在控制電路上。

單晶片微電腦的優點及特性：

1.體積小，成本低，容易維修。

單晶片微電腦將CPU ，記憶體，I/O 製作在同一晶片上，體積明顯比一般微電腦縮小。

2.硬體接線容易。

單晶片在使用時只需接少許元件即可做廣泛的控制，除了提高硬體接線的可靠度外，在裝配和維護上也相當容易。

3.架構簡單。

單晶片的硬體架構簡單，指令也較少，容易學習和撰寫。

4.擴充性佳。

由於HMOS 和CMOS 的發展，使單晶片不論在功能，包裝密度上都增強許多，同時單晶片提供READ 和WRITE 等控制信號，在外部記憶體和I/O 的擴充上也相當方便。

第三章 MCS--51 單晶片介紹

3.1 MCS-51 簡介

MCS-51 是Intel 所生產的8051 系列單晶片的總稱，這家族中還有8051 ，8751 ，8031 ，8032 ，8052 ，80C51 等。這些單晶片雖然在編號上有所不同，卻都是使用相同的CPU ，指令集，只是附加的周邊上有所不同，例如8751 是內含4k ERROM 的MCS-51 版本。

MCS-51 是在1980 ，由Intel 根據MCS-48 的架構，所發展出功能更強，速度更快的單晶片。Intel 發展初期，是以HMOS的製程發展而成，經過數年，又改以CMOS 生產，例如8051 就是此款。目前除了Intel 生產之外，也有好幾家公司製造，功能上也有增加，例如AMD ，PHILPS ，ATMEL ，DALLS 等，其中又以ATMEL 製造的AT89C51 不需紫外線清洗，而是電子式抹除，十分方便。

8051 的主要功能列舉如下：

1.為一般控制應用的8 位元單晶片。

2.晶片內部具時脈震盪器(最高工作可至12MHZ)。

3.內部程式記憶體(ROM)為4k 位元組。

4.內部資料記憶體(RAM)為128k 位元組。

5.外部程式記憶體可擴充至64k 位元組。

6.外部資料記憶體可擴充至64k 位元組。

7.32 條雙向輸入輸出線，且每條均可以單獨做I/O 控制。

8.5 個中斷向量元。

9.2 組獨立的16 位元定時器。

10.1 個全多工串列通訊埠UART(通用非同步接收和傳送9
器)。

11.單晶片提供位元邏輯運算指令。

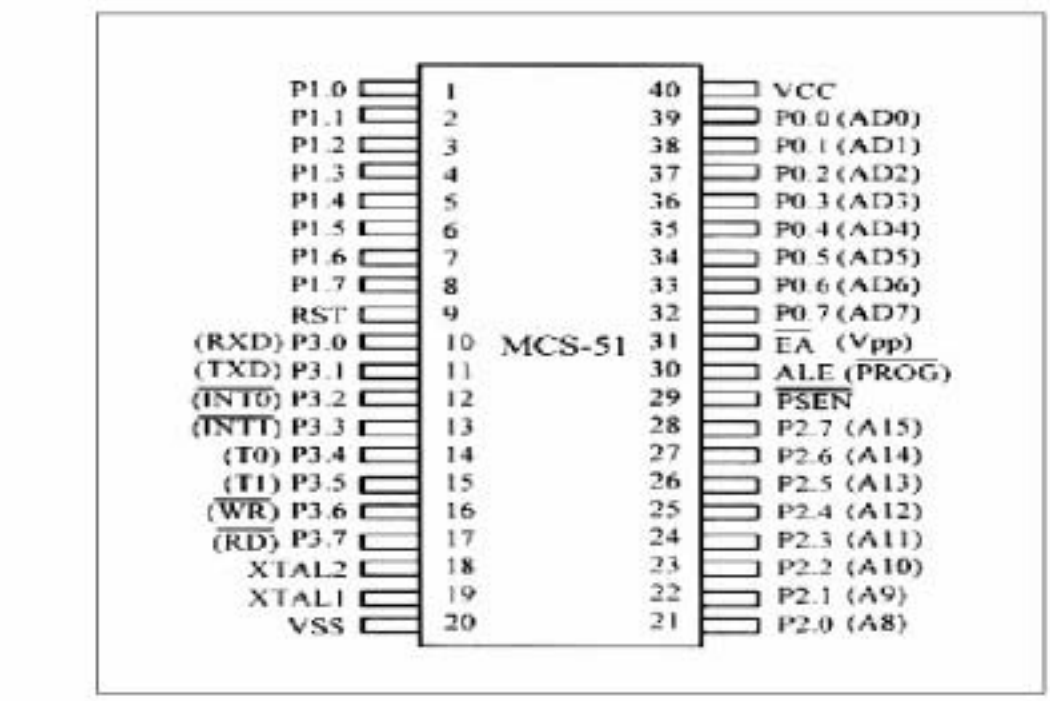


圖3-1 8051 單晶片的IC 接腳圖

8051 接腳說明

接腳	名稱	功用
1~8	P1.0~P1.7	埠 1(P1)是內部已經具有提升電阻的 8 位元雙向 I/O 埠腳，第 1 腳 P1.0 是 LSB，第 8 腳 P1.7 是 MSB。
9	RESET	8051 的重置(RESET)腳，平常 8051 工作

<table>
<tr><td></td><td></td><td>時這隻腳需要保持 LOW 的狀態，當這隻腳由外部輸入 HIGH 的信號時，8051 將被重置，重置後的 8051 將由位址 0000H 開始執行。內部工作暫存器 RAM00H~7FH 的內容不受影響，但特殊暫存器(SFR)將重新設定爲初設值。</td></tr>
<tr><td>10~17</td><td>P3.0~P3.7</td><td>P3 也是內部已經具有提升電阻的 8 位元雙向 I/O 埠腳，另外它的每一隻 I/O 腳除了可以當作單純的輸入/輸出外，也兼具第 2 功能。
<table>
<tr><td>P3.0</td><td>RXD 串列埠輸入端子</td></tr>
<tr><td>P3.1</td><td>TXD 串列埠輸出端子</td></tr>
<tr><td>P3.2</td><td>INT0 外部中斷 0 輸入端</td></tr>
<tr><td>P3.3</td><td>INT1 外部中斷 1 輸入端</td></tr>
<tr><td>P3.4</td><td>T0 計時/計數器 0 外部輸入端</td></tr>
<tr><td>P3.5</td><td>T1 計時/計數器 1 外部輸入端</td></tr>
<tr><td>P3.6</td><td>WR 外部資料記憶體 RAM，寫入之激發脈波輸出端</td></tr>
<tr><td>P3.7</td><td>RD 外部資料記憶體 RAM，讀出之激發脈波輸出端</td></tr>
</table></td></tr>
<tr><td>18~19</td><td>XTAL2(18)
XTAL1(19)</td><td>這兩隻腳是 8051 外部時脈震盪器的輸入端，若在這兩隻腳的兩端並聯一個 12MHZ 晶體振盪器，則 8051 內部就會產生 12MHZ 的工作頻率，供內部使用</td></tr>
</table>

20	VSS	這隻腳是 8051 的接地腳，使用 8051 時此腳必須與系統的地線接在一起
21~28	P2.0~P2.7	P2 也是內部已經具有提升電阻的 8 位元雙向 I/O 埠腳，P2 除了當一般的雙向 I/O 埠腳以外，如果在 8051 外部擴充有程式記憶體 EPROM，或資料記憶體 RAM 時，P2 就變成 8051 匯流排的高位元組與 P0 的低位元組共同組成 16 位元的位址匯流排，對外存取外部記憶體.
29	$\overline{\text{PSEN}}$	讀取外部程式記憶體的致能信號，是 8051 用來讀取存放在外部程式記憶體的程式時，所用來讀取信號的輸出端，它通常與外部程式記憶體的 OE 腳連接。8051 就是利用 PSEN 與 RD 分別致能外部的程式記憶體與資料記憶體，如此，對外擴充的最大記憶體可達 64K byte 的 EPROM 及 64K byte 的 RAM
30	ALE	位址栓鎖致能腳，8051 可以使用這隻腳發出信號，將 P0 上的位址匯流排信號(A0~A7)拴鎖在拴鎖器中，使 P0 具有資料位址匯流排的能力，通常拴鎖器例如 74LS373 等
31	$\overline{\text{EA}}$	擴充內部記憶體選擇，當 EA=1 時 CPU 執行內部程式記憶體，EA=0 時執行外部程式記憶體，因此，如果使用 8751 或 8051 的內部程式記憶體，則必須將 EA 接 VCC

32~39	P0.0~P0.7	P0 是 8 位元的開及設極式雙向 I/O 埠，當你是寫 1 到 P0 時，則 P0 將成爲浮接(float)狀態，亦可視爲高抗阻狀態，但是 P1，P2，P3 因爲內部都已具備提升(pull-up)電阻的功能，因此並不會造成類似狀態，故若要將 P0 當做一般 I/O 使用時，其埠腳應於外端再接一提升電阻 4.7kΩ，使其輸出的”H”準電壓得以有栓鎖的功能，不致造成浮接。P0 還有另一項功能，就是在外面擴充程式記憶體時，P0 就當作位址匯流排(A0~A7)和資料匯流排(D0~D7)多工使用，要成此功能必須在 P0 的外部加一個 8 位元的位址栓鎖器。此 A0~A7 則與 P2 所提供的 A8~A15 組合成一個 16 位元的 address bus。因此 8051 可在外部擴充成 64k 的記憶體
40	VCC	8051 的電源輸入端

表3-1 MCS-51 接腳功能表

3.2 MCS-51 硬體架構

在一顆MCS-51 單晶片中，其功能跟構造可簡化成圖3-2 所示:

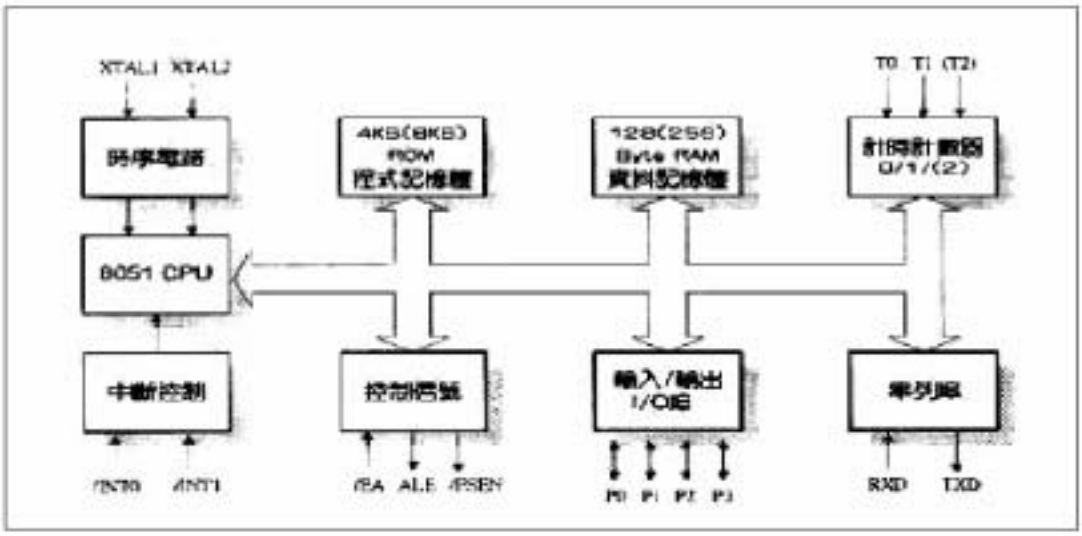


圖3-2 MCS-51 內部構造方塊圖

1.震盪與時序方塊

MCS-51 內部有時脈震盪電路，只要外部加上石英震盪晶體，即可產生脈波頻率非常穩定的脈波信號，所有MCS-51 單晶片的時序都是以此為基準。

2.CPU 方塊

這是整個單晶片的控制處理中心，CPU 讀取位於程式記憶體(ROM 或EPROM)程式碼，經過計算及處理後，將結果送至各個暫存器或輸出入埠上，並且接受內部和外部的中斷信號，然後執行中斷服務程式。只要電源加入且震盪器開始動作後，CPU 就會開始不停的動作。

3.程式記憶體

MCS-51系列單晶片，8051/8751提供內部4096Bytes(4Kbytes)的程式記憶區，專供程式儲存指令碼的地方。CPU 所執行的程式指令，即是到這裡來提取。8052 提供內部8192Bytes(8Kbytes)的程式記憶區，而8031 則不提供此方塊。

若有內部程式記憶區時，CPU 可以選擇執行的程式指令，是由內部的程式區提取或由外部的程式區提取。程式區的內容，只能讀出但不能寫入。

4.資料記憶體

MCS-51 系列中的8051/8031 及8751 單晶片都提供有128個Bytes 的可讀/寫資料記憶區，而8052 系列則有256 個Bytes的資料記憶區。這資料區中有16 個Bytes 共128bits的區域是可直接做單一位元定址(Bit Adress)的，同時MCS-51 也提供相當好用的位元處理指令。

5.四組可規劃輸入/輸出埠P0 ，P1 ，P2 ，P3

這四個埠共提供4X8=32 條I/O 線，所有的埠都可以做位元組輸出入埠(Byte I/O)或做單一位元輸出入埠(bit I/O)，當MCS-51做外部記憶體擴充時，必須用PORT0 ，PORT2 當作資料/位址線，配合ALE ，/PESN ，及/WR ，/RD 等控制線產生必要的控制信號，作讀出及寫入信號。

6.計時/計數方塊

MCS-51 系列的單晶片均有2 個16 位元的計時/計數器，而8052則有3 個。每個計時/計數器有多種模式供選擇。

7.可規劃I/O 串列方塊

MCS-51 單晶片可透過此串列輸出入埠介面，與外界的電腦或儀器設備做資訊的交換，也可透過此介面做I/O 的擴充。

第四章 89C51

4-1:89C51 介紹

89C51 單晶片微電腦是一個40隻接腳的大型積體電路（VLSI），接腳
的排列圖：

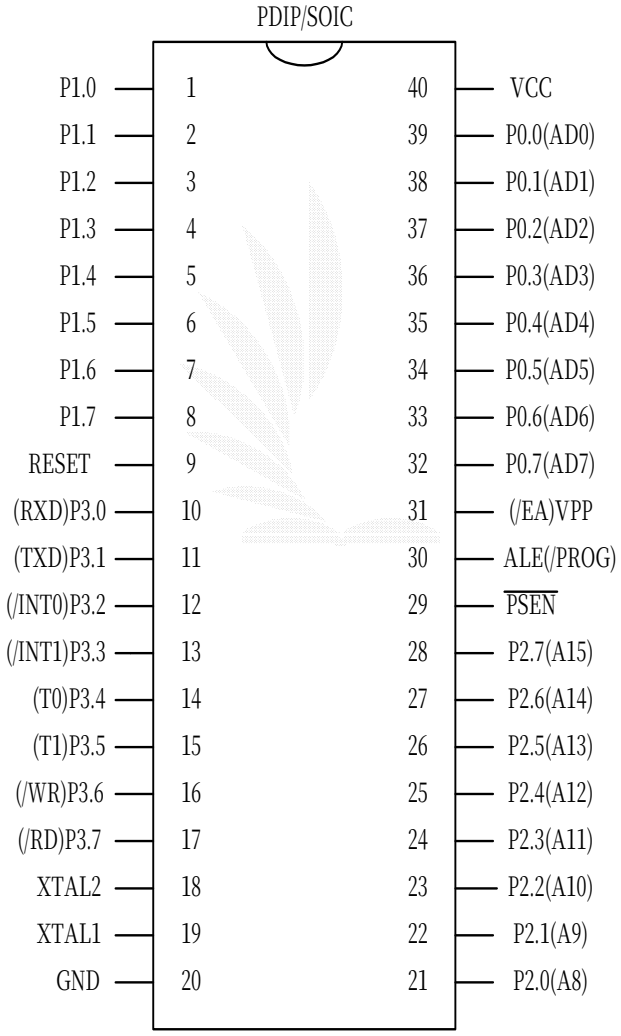


圖 4-1　89C51 接腳功能&外形圖

4-2:89C51 特點

１.最適合控制應用的８位元ＣＰＵ。

２.具布林代數處理（單位元邏輯）能力。

３.６４ＫＢｙｔｅｓ的程式記憶體位址空間（ＲＯＭ)。

４.６４ＫＢｙｔｅｓ的資料記憶體位址空間（ＲＡＭ)。

５.晶片內部有４ＫＢｙｔｅｓ的程式記憶體。

６.晶片內部有１２８Ｂｙｔｅｓ的資料記憶體。

７.３２個可雙向和獨立定址的Ｉ／Ｏ線。

８.兩組１６位元計時器／計數器。

９.一組全雙工ＵＡＲＴ（通用非同步接收和傳送器)。

１０.具兩層優先權的５個中斷源結構。

１１.晶片內部有時脈振盪器（最高為１２ＭＨｚ)。

１２.具有４ＫＢｙｔｅｓ的快速可規劃和可抹除的ＲＯＭ可寫入／抹除１０００次以上， 程式可保存１０年以上。

１３.操作電壓範圍：Ｖｃｃ＝５Ｖ±１０％或２０％。

１４.操作頻率：０Ｈｚ　２４ＭＨｚ。

１５.三個可規劃的記憶體上鎖位元。

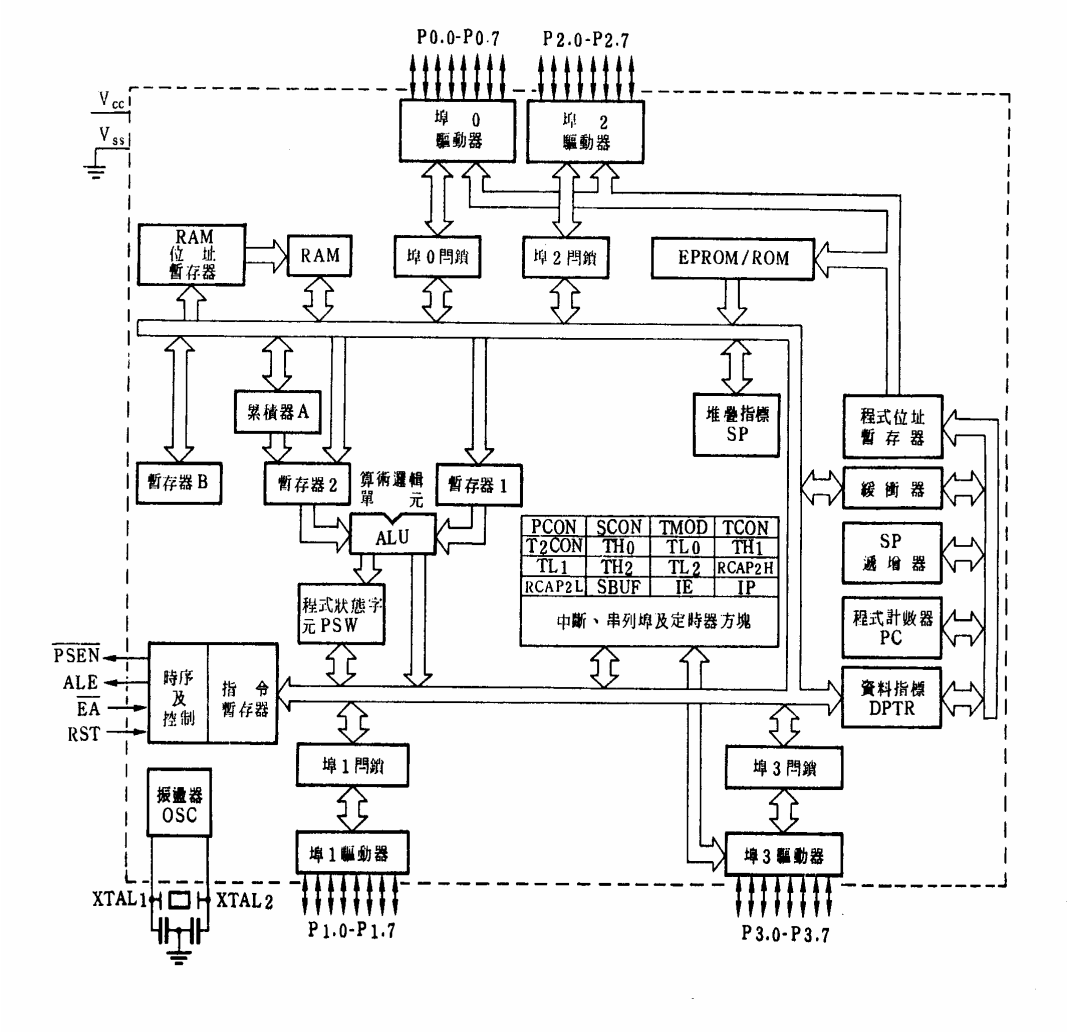


圖 4-2 89C51 內部結構方塊圖

4-3:89C51 接腳功能說明

Vss ：

(1) 第 20 腳。

(2) 電路之地電位腳。

Vcc ：

(1) 第 40 腳。

(2)電源接腳。

XTAL1 及 XTAL2 ：

(1)第 19 腳及第 18 腳。

(2) 兩腳之間需接一個 3.5　12MHZ 之石英晶體。

(3) 請參考圖 4-3-1

(4) 常用之石英晶體有 3.58MHZ、6MHZ、11.0592MHZ 12MHZ。

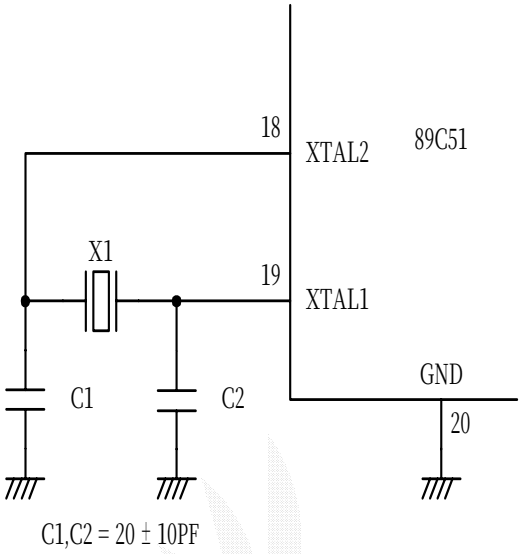


圖 4-3-1 振盪電路

RESET ：

(1)　第 9 腳。重置輸入腳。

(2) 此腳內部已有一個 50ｋΩ　300ｋΩ的電阻器接地，所以只需接一個電容器至+VCC 即可在電源 ON 時產生開機重置的功能。

(3) 如有需要, 亦可在電容器兩端並聯一個常開按鈕, 以便壓此按鈕時可強迫系統重置。

(4) 請參考圖 4-3-2。

(5) 當重置信號發生後會產生下列作用：

a．重置特殊功能暫存器的值。

b．所有的埠(Port1，Port3)都成為輸入狀態。

c．令 CPU 從位址 0000H 開始執行程式。

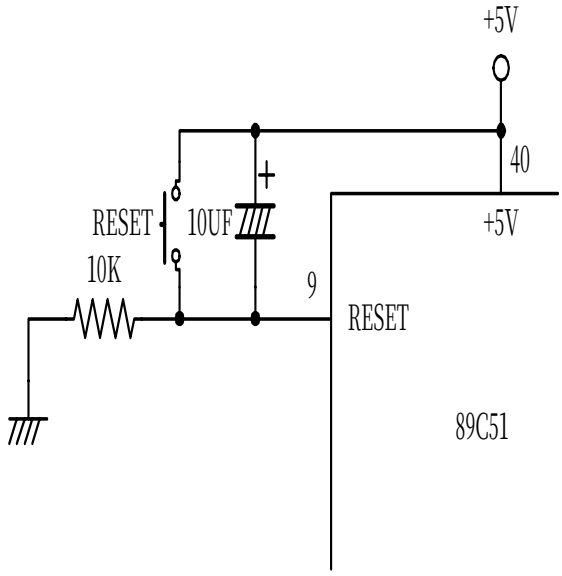


圖 4-3-2 開機重置電路

暫存器名稱	重置值
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0～P3	FFH
IP(8051)	×××00000B
IP(8052)	××000000B
IE(8051)	0××00000B
IE(8052)	0×000000B
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2(8052)	00H
TL2(8052)	00H
RCAP2H(8052)	00H
RCAP2L(8052)	00H
SCON	00H
SBUF	不一定
PCON(HMOS)	0×××××××B
PCON(CHMOS)	0×××000 0B

註：(8051)代表8031、8051、8751等編號。
(8052)代表8032、8052、8752等編號。
(HMOS)表示HMOS版本。
(CHMOS)表示CHMOS版本。

表 4-3 特殊功能暫存器重置後之預設值

/ＥＡ ：

(1)第 31 腳。

(2)當 /EA 腳接地時,內部程式記憶體失效,CPU 被迫只讀取外部的程式記憶體。

(3)8051、8052、8751、8752 等編號,此腳必需接至+Vcc。

(4)8031、8032 等編號,此腳必需接地。

P0.0 P0.7:

(1) 第 32 39 腳。

(2) 8 位元之輸入/輸出埠。稱為 Port0,簡稱為 P0。

(3) 每隻腳均可當成輸入或輸出腳用。

(4) 接腳 P0.0 P0.7 均為開汲極(open drain) 結構,若欲輸出 Hi 或 Low 之電壓需自己在接腳上外部提升電阻(external pull-up),請參考圖 4-3-3。

(5) 當外接記憶體或外接 I/O 時,必須利用 P0.0 P0.7 作為位址匯流排及資料匯流排。

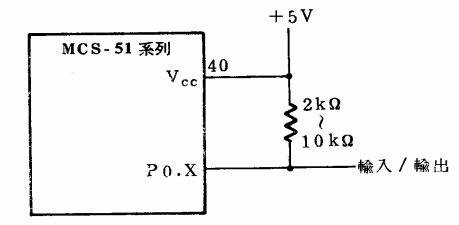


圖 4-3-3 P0.0 P0.7 任一腳接上外部提升電阻器的方法

(6) Port 0 做輸出埠用時,每隻接腳均可沉入(sink) 8 個 LS TTL 負載。

(7) 若某接腳欲當作輸入腳用,則需先將 1 寫入這隻接腳。

P1.0 P1.7：

(1) 第 1 第 8 腳。

(2) 8 位元之輸入/輸出埠。稱為 Port1，簡稱為 P1。

(3) Port 1 為具有內部提升電阻(約 30kΩ)的雙向輸入/輸出埠。可以驅動 4 個 LS TTL 負載。

(4) 每隻腳均可當成輸入腳或輸出腳使用。

(5) 若某接腳欲當成輸入腳用，則需先將 1 寫入這隻腳。

P2.0 P2.7：

(1) 第 21 第 28 腳。

(2) 8 位元之輸入/輸出埠。稱為 Port 2,簡稱為 P2。

(3) Port 2 是具有內部提升電阻器(約 30kΩ)的雙向輸入/輸出埠。可以驅動 4 個 LS TTL 負載。

(4) 每隻腳均可當成輸入腳或輸出腳用。

(5) 若某接腳欲當作輸入腳用,則需先將 1 寫入這隻接腳。

(6) 當 CPU 使用 16 位元的位址對外部記憶體進行存取時 Port 2 被用來輸出位址的高位元組。

P3.0 P3.7：

(1) 第 10 第 17 腳。

(2) 8 位元之輸入/輸出埠。稱為 Port3，簡稱為 P3。

(3) Port 2 是具有內部提升電阻器的雙向輸入/輸出埠。

(4) 可以驅動 4 個 LS TTL 負載。

(5) 每隻腳均可當成輸入腳或輸出腳使用。

(6) 某接腳欲當成輸入腳用, 必需先將 1 寫入這隻腳。

(7) Port3 的接腳可以作為下列特殊用途

接腳名稱	特　殊　功　能
P3.0	RXD(串列埠的輸入腳)
P3.1	TXD(串列埠的輸出腳)
P3.2	/INT0(外部中斷 0 的輸入腳)
P3.3	/INT1(外部中斷 1 的輸入腳)
P3.4	T0(計數器 0 的輸入腳)
P3.5	T1(計數器 1 的輸入腳) /WR(當 CPU 欲資料送至外部 RAM 或外部
P3.6	I/O 裝置時,此腳會產生負脈波。稱爲 寫入脈波輸出腳。) /RD(當 CPU 欲從外部 RAM 或外部 I/O 讀
P3.7	取資料時,此腳會產生負脈波。稱爲 讀取脈波輸出腳。)

ＡＬＥ：

(1)第 30 腳。位址閂鎖致能(address latch enable)輸出腳。

(2)當 CPU 對外部裝置存取資料時,此腳輸出脈波之負緣可用來鎖住(latch)由 Port 0 送出之低位元組位址。

/ＰＳＥＮ：

(1)第 29 腳。外部程式記憶體致能(program store enable)輸出腳。

(2) 當 CPU 欲讀取外部程式記憶體的內容時,此腳會自動產生負脈波。

第五章 LCD 簡介

5-1 LCD 簡介

液晶顯示器(Liquid Crystal Display)為目前使用最廣泛的顯示裝置之一，諸如計算機、電子儀器、事務機器、電器產品、筆記型電腦等。LCD 能顯示大小寫英文字、數字、日文字、與特殊符號等各種字型，LCD 本身不發光必須藉由外界光的反射才能看見圖象，與LED 顯示原理不同，LED 本身即具有發光的能力，所以在夜間使用時，需要在LCD 背面加裝光源，稱為背光。但是LCD 顯示器的功率消耗很小，而且以低電壓驅動，所以非常省電。LCD 介面電路方塊如圖5-1 。

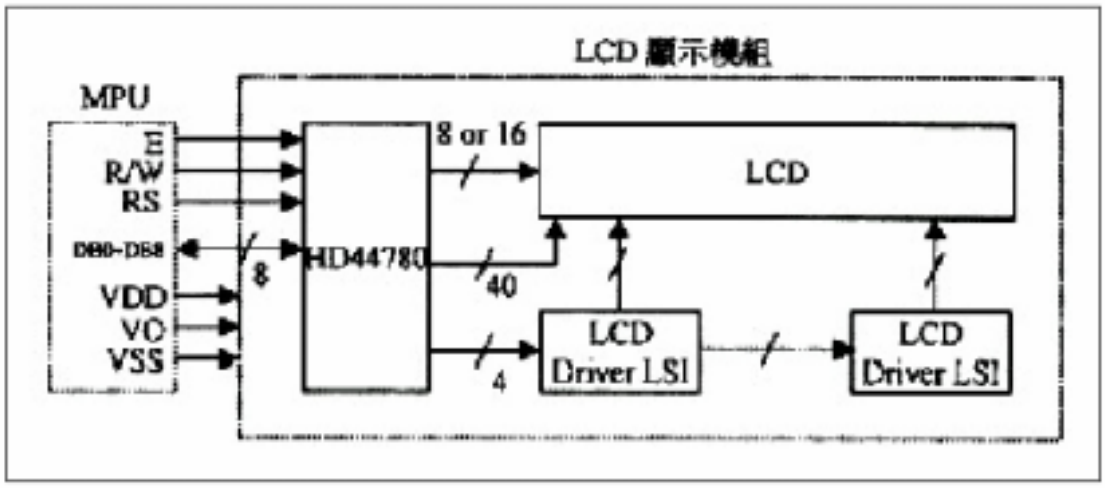


圖5-1 LCD 介面電路方塊圖

5-2 LCD 內部結構

1.顯示資料記憶體(DD RAM)有80 個位址，HD44780 晶片最多可同時顯示80 個字型。各種LCD 模組位址對映如表5-1 。

00H	01H	02H	03H	…	24H	25H	26H	27H
40H	41H	42H	43H	…	64H	65H	66H	67H

表 5-1a　40X2　LCD 模組位址

00H	01H	02H	03H	…	10H	11H	12H	13H
40H	41H	42H	43H	…	50H	51H	52H	53H

表 5-1b　20X2　LCD 模組位址

00H	01H	02H	03H	…	0CH	0DH	0EH	0FH
40H	41H	42H	43H	…	4CH	4DH	4EH	4FH

表 5-1c　16X2　LCD 模組位址

2.字元產生器(CG RAM)，可由使用者自行設計8 個5X7 字型，每一個5X7 自行需要8 個位元組，所以CG RAM 共有64 個位元組，其位址為(00H~3FH)。與字型碼對映如表5-2

字型碼		CG RAM 位址	
2 進制	16 進制	2 進制	16 進制
0000X000	00H(08H)	XX000000	00H
		XX000001	01H
		XX000010	02H
		XX000011	03H
		XX000100	04H
		XX000101	05H
		XX000110	06H

		XX000111	07H
0000X001	01H(09H)	XX001000	08H
		XX001001	09H
		XX001010	0AH
		XX001011	0BH
		XX001100	0CH
		XX001101	0DH
		XX001110	0EH
		XX001111	0FH
: :	: :	: :	: :
0000X111	07H(0FH)	XX111000	38H
		XX111001	39H
		XX111010	3AH
		XX111011	3BH
		XX111100	3CH
		XX111101	3DH
		XX111110	3EH
		XX111111	3FH

表5-2 字元產生器(CG RAM)與字型碼對映

3.字元產生器(CG ROM)，在HD44780 晶片內部有192 個5X7 字型的ROM ，如表5-3 。使用者不可以更改其內容，只要將22 字型碼寫入DD RAM 中，即可在LCD 指定位址上顯示其字型。

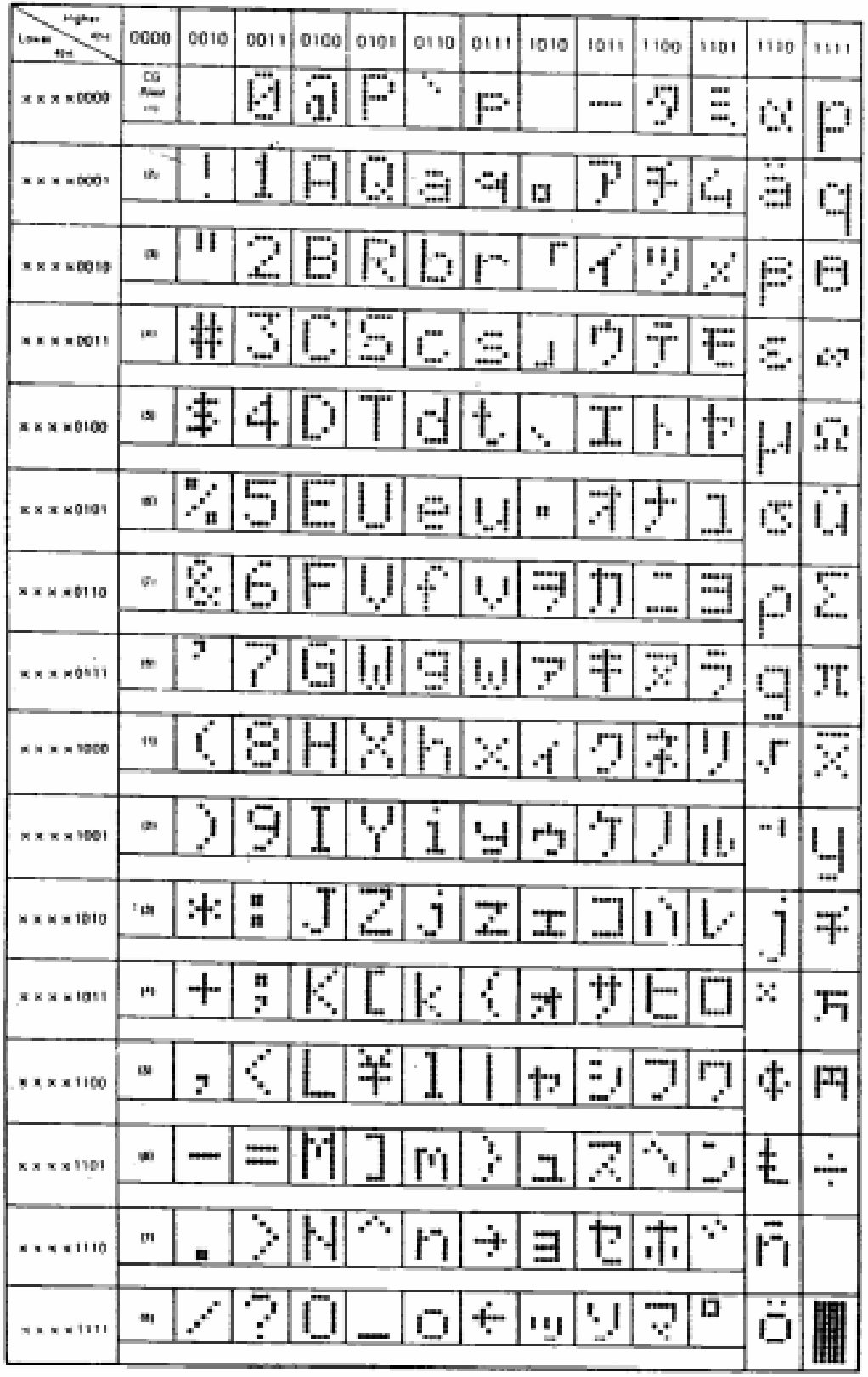


表5-3 LCD 內建字型表23

5-3 LCD 接腳規格

雖然LCD 的接腳因製造廠商的不同，而有不同的排列方式，不過都是編號1 至14 的14 根接腳。

腳位	符號	輸入/輸出(I/O)	功能
1	VSS	I	接地腳
2	VDD	I	+5V 電源
3	VO	I	顯示明暗對比控制
4	RS	I	RS=0，選擇指令暫存器 RS=1，選擇資料暫存器
5	R/W	I	R/W=0，將資料寫入 LCD R/W=1，自 LCD 讀取資料
6	E	I	致能
7	DB0	I/O	資料匯流排(LSB)
8	DB1	I/O	資料匯流排
9	DB2	I/O	資料匯流排
10	DB3	I/O	資料匯流排
11	DB4	I/O	資料匯流排
12	DB5	I/O	資料匯流排
13	DB6	I/O	資料匯流排
14	DB7	I/O	資料匯流排(MSB)

表5-4 LCD 接腳說明

註：有些LCD 模式之VDD 與VSS 接腳相反，使用前先測試，將VO 與VSS 接腳短路接地，VDD 接+5V 正電源，此時LCD 應會亮，如果不亮，將正負反接即可。

5-4 LCD 內部旗號與暫存器

1.忙碌旗標 Busy Flag(BF)BF 旗標是LCD 用來告訴CPU 其內部是否忙碌中的一個旗標，若BF=1 ，表示LCD 正在處理內部的工作，所以在此時不能將資料寫入LCD ，當BF=0 ，表示CPU 可將資料寫入LCD 。當接腳RS=0 且R/W=1 時，忙碌旗標會由DB7輸出。

2.暫存器

LCD 模組內只有2 組8 位元暫存器，稱為指令暫存器(Instruction Register)和資料暫存器(Data Register)，它們都是

8 位元暫存器，由RS 腳來選用。

E	RW	RS	功能說明
0	X	X	不動作
1	0	0	將指令寫入 LCD 指令暫存器 IR
1	0	1	將資料寫入 LCD 之 RAM 中
1	1	0	由 LCD 之資料暫存器 DR 讀取資料
1	1	1	讀取忙碌旗標 BF 及位址計數器 AC 之內容，其中 DB7=BF，DB6~DB0=AC

表5-5 控制腳功能

指令暫存器IR 用來接收單晶片送來的命令，例如清除顯示，功能設定等等，資料暫存器DR 則用來接收單晶片要寫到DD RAM(共80Byte)或CG RAM(共64Byte)的資料緩衝區。當單晶片寫到DR 暫存器之後，LCD 內部的控制電路會將資料自動寫到DD RAM 或CG RAM 中，而位址是由LCD 內部的位址計數器(Address Counter)所指定，而單晶片要讀取資料時，需先將欲讀取的位址放入IR 暫存器中，LCD 就會將其內容放入DR 中，然後單晶片就可以去讀取DR 的資料。

3.位址計數器AC

位址計數器是用來指定欲存取的DD RAM 和CG RAM 的位址，設定指令將位址寫入IR 暫存器之後，LCD 內部控制電路會將IR 暫存器的內容送至AC ，當資料存取之後，AC 便會自動加一或減一。

4.字元產生器Character Generate ROM(CG ROM)

LCD 內部有一個存放字型的ROM ，裡面存著192 個5X7點矩陣的字形，這些字型由存放在DD RAM 中的ASCII碼叫出來顯示，例如：”A ”的字型碼為01000001(即41H)。

5.自創字型產生器

LCD 模組除了提供標準的字型CG ROM 供使用者使用之外，還提供了一塊64 位元組的CG RAM 空間，給使用者存放自己設計的字型。一個字要8 個位元組，因此最多可存放8 個字元。要顯示出自己的字元時，須先將5X7點矩陣圖形放入CG RAM 中，而要叫出字元時，其字型
碼為00H~07H 。

5-5 LCD 模組控制指令

LCD 模組可以接受CPU 送至IR 暫存器的命令，並加以執行，其指令共有11 道，除了清除顯示和游標歸位兩道指令的執行時間為1 。64ms 外，其餘皆為40us 。

1.清除顯示

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	0	0	0	0	0	0	0	0	0	1

將DD RAM 的資料全部填入20H(空白字)，並將游標移到左上角HOME 的位置，清除位址計數器AC=0 。執行時間約1.64ms 。

2.游標歸位　　RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

0 0 0 0 0 0 0 0 1 X

DD RAM 的資料保持不變，僅將游標移至左上角HOME位置，清除位址計數器AC=0 。執行時間約1.64ms 。

3.輸入模式設定　RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

0 0 0 0 0 0 0 0 I/D S27

S	I/D	動作說明
0	0	顯示字元不動，游標左移，AC 值減1
0	1	顯示字元不動，游標右移，AC 值加1
1	0	游標不動，顯示字元右移，AC 值不變
1	1	游標不動，顯示字元左移，AC 值不變

4.顯示器控制

RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

0 0 0 0 0 0 1 D C B

D ：顯示器顯示(Display)控制位元，D=0 ，關閉，D=1 ，開啟。

C ：游標(Cursor)顯示控制位元，C=0 ，不顯示，C=1 ，顯示。

B ：游標閃爍(Blink)控制位元，B=0 ，不閃爍，B=1 ，閃爍。

5.游標移位控制

RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

0 0 0 0 0 1 S/C R/L X X

S/C	R/L	動作說明
0	0	游標左移，AC 值減1
0	1	游標右移，AC 值加1
1	0	整個顯示幕向左移動
1	1	整個顯示幕向右移動

本命令不改變DD RAM 資料，即不做資料讀寫動作，而僅移動游標或整個顯示幕，與輸入模式設定不同，輸入模式設定是設定在每次讀寫DD RAM 時控制AC 或顯示字元的移動情形。

6.功能設定

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	X	X

DL ：資料長度設定，DL=0 ，4 位元(DB7~DB4)，DL=1 ，8 位元。

N ：列數設定，N=0 ，1 列顯示，N=1 ，2 列顯示。

F ：字型設定，F=0 ，5X7 點矩陣字型，F=1 ，5X10 點矩陣字型。

7.CG RAM 位址設定

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	A5	A4	A3	A2	A1	A0

設定CG RAM 的位址，由A5~A0 之6 個位元定址00H~3FH 位址共64 個位元組，當寫入本命令之後，接著輸入的資料將寫入到CG RAM 中。

8.DD RAM 位址設定

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A6	A5	A4	A3	A2	A1	A0

設定DD RAM 的位址，由A6~A0 之7 個位元定址，其位址如表5-1 ，當寫入命令之後，接著輸入的資料將寫入到DD RAM 中。

9.讀取忙碌旗標BF 及位址計數器AC 內容

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	A6	A5	A4	A3	A2	A1	A0

當BF=1 時，表示LCD 模組正在處理資料，不可以在寫入資料。而AC 值為最近設定的RAM 位址(CG RAM 或DD RAM)。

10.將資料寫入DD RAM 或CG RAM 中

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

11.自DD RAM 或CG RAM 讀取資料

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

LCD 模組對DD RAM 或CG RAM 寫入或讀取資料之前，必須先設定DD RAM 或CG RAM 位址，如果設定DD RAM 位址，則下一指令即為讀取或寫入資料到DD RAM 中，反之，如果設定CG RAM 位址，則下一指令即讀取或寫入到CG RAM 中。

命令 (Instruction)	指令碼 RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	功能說明
清除顯示器	0	0	0	0	0	0	0	0	0	1	將所有顯示的資料清除，且將游標回到原點（位址 00H）
游標歸位	0	0	0	0	0	0	0	0	1	*	將游標回到原點，但 DD RAM 的內容不變
進入模式設定	0	0	0	0	0	0	0	1	I/D	S	設定游標移動方向，或顯示資料後是否全移位址是否閃動 (I)
顯示器 ON/OFF 控制	0	0	0	0	0	0	1	D	C	B	控制所有顯示器 (D)，游標 (C) 的 ON/OFF，和游標位址是否閃動 (B)
顯示器或游標移動方式設定	0	0	0	0	0	1	S/C	R/L	*	*	在不需要改變 DD RAM 內容的情況下，移動游標和顯示器資料
功能設定	0	0	0	0	1	DL	N	F	*	*	設定界面資訊排寬度 (DL)，顯示器行數 (N)，和字形 (F)
CG RAM 位址設定	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	設定 CG RAM 位址，以後輸入或讀出 CG RAM 的資料
DD RAM 位址設定	0	0	1	AD6	AD5	AD4	AD3	AD2	AD1	AD0	設定 DD RAM 位址，以後輸入或讀出 DD RAM 的資料
比較旗號 /位址讀取	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	讀出 LCD 內部的 BF 旗號和位址計數器內容
資料寫到 CG RAM 或 DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	將資料寫到 DD RAM 或 CG RAM
從 CG RAM 或 DD RAM 讀出資料	1	1	D7	D6	D5	D4	D3	D2	D1	D0	從 CG RAM 或 DD RAM 讀取資料

*:Don't care bit

I/D=1:遞加
I/D=0:遞減

S=1:顯示器移動
S=0:顯示器不移動

D=1:要顯示
D=0:不顯示

BF=1:LCD 內部正在動作
BF=0:LCD 準備好接受資料

C=1:游標 ON
C=0:游標 OFF

B=1:閃爍
B=0:不閃爍

S/C=1:顯示器移動
S/C=0:游標移動

F=1:5×10 點矩陣
F=0:5×7 點矩陣

R/L=1:向右移動
R/L=0:向左移動

DL=1:8 位元
DL=0:4 位元

N=1:二行
N=0:一行

表5-6 LCD 控制指令表32

5-6 LCD 初始化流程

LCD 模組在送電後，單晶片必須規劃LCD 的各項功能或工作模式，LCD 才能正常工作，這個動作稱為LCD 初始化。

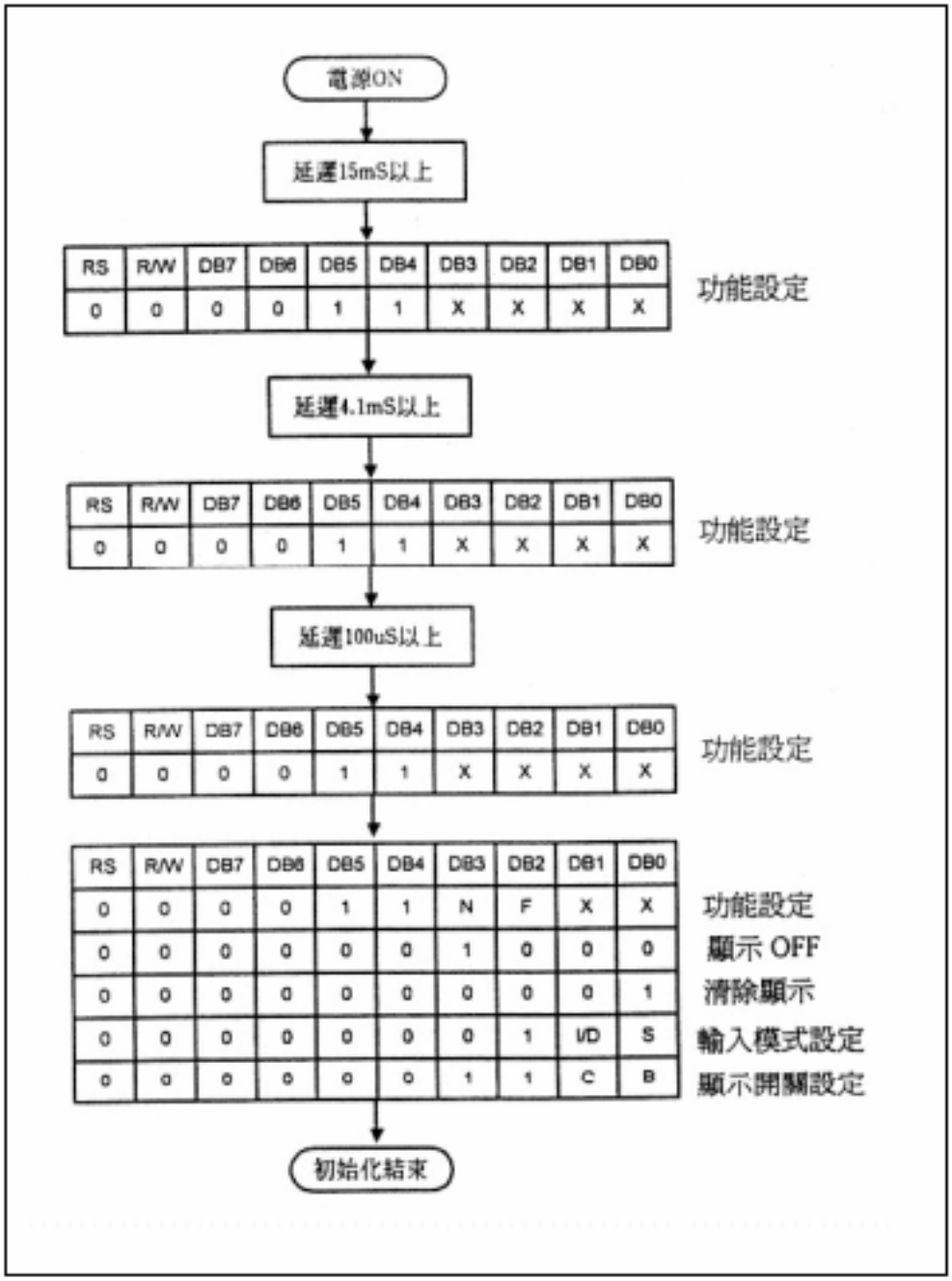


圖5-2 8 位元介面初始化流程圖33

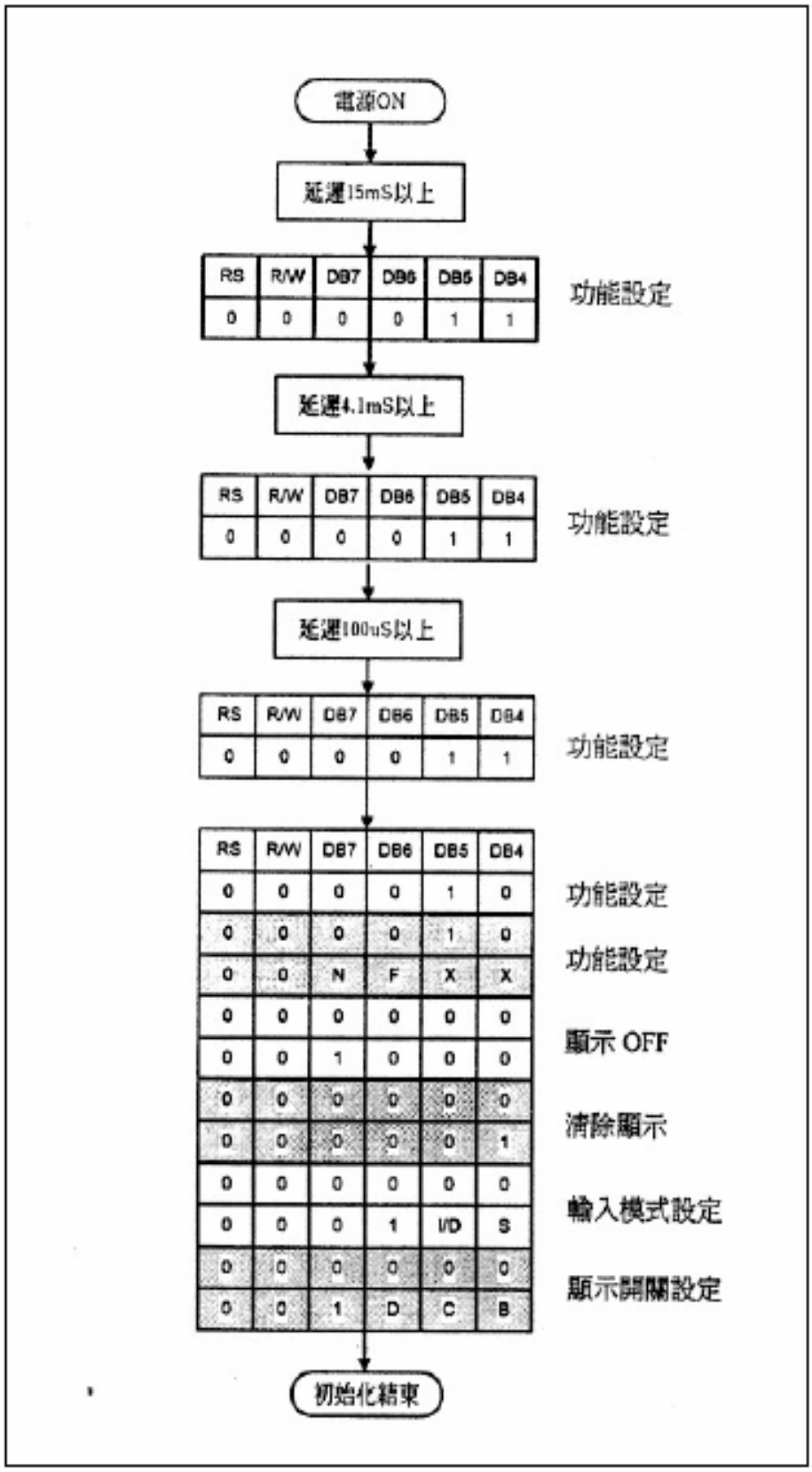


圖5-3 4 位元介面初始化流程圖34

第六章 系統設計說明

6-1 系統設計說明

本次專題中，主要將賣場置物櫃中機械式的密碼鎖以電子電路來實現他。利用 89C51 進行密碼的比對及改變使用者的密碼。賣場置物櫃中的密碼鎖只有四碼可以由使用者改變，些專題中則可以由使用者自行決定密碼的長度（最多十六個位元）及更換密碼，為其一大特色。

在硬體方面，利用 89C51 為主，並配合矩陣式鍵盤及 LCD 顯示裝置。矩陣式鍵盤可以減少 89C51 的 IO 使用量，在些設計的矩陣鍵盤為 4X4。LCD 顯示裝置主要用來顯示系統的操作訊息。

在軟體方面主要使用 PE2 或 HE4 來編輯系統程式，程式主要是利用組合語言編輯，再將程式存成＊.asm，再經由 EP51 進行程式的組譯，並檢查程式碼是否有錯誤產生。程式經由 EP51 組譯後，會產生＊.obj 檔（如無錯誤），再經由 ELINK 執行檔將程式做連結的動作，並產生＊.hex 檔。如下圖 6-1 為程式發展的流程圖。

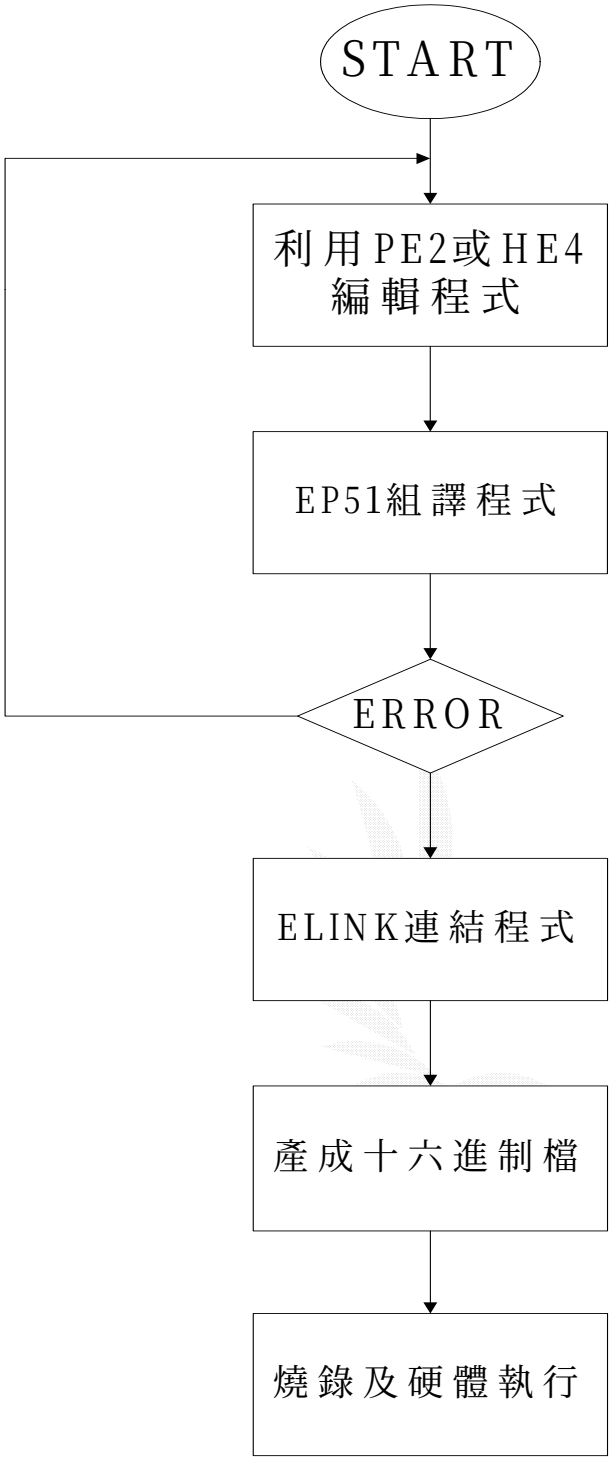

START
利用PE2或HE4
編輯程式
EP51組譯程式
ERROR
ELINK連結程式
產成十六進制檔
燒錄及硬體執行

圖 6-1 程式發展流程圖

6-2 硬體電路

6-2-1 矩陣鍵盤電路

本節中主要是利用矩陣鍵盤來減少 89C51 的 IO 使用量，或是在硬體電路上加入編碼 IC 也可以完成鍵盤電路，以減少副程式的編輯；或是寫一個鍵盤掃瞄副程式來控制系統的輸入裝置。上述的兩種方法皆有其利弊，但皆是值得學習的設計方法。如圖 6-3 所示，為一個矩陣鍵盤電路圖。

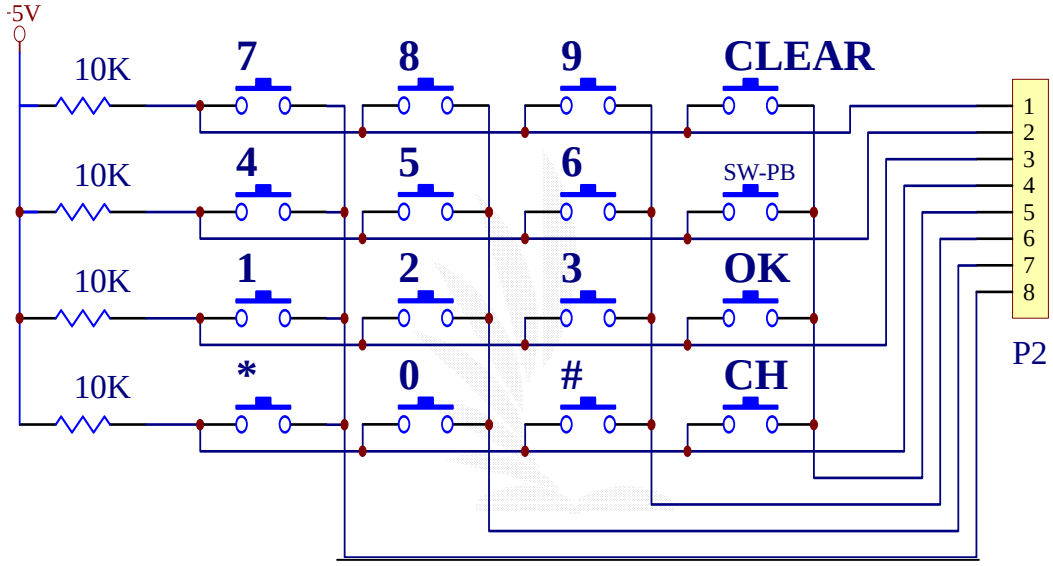
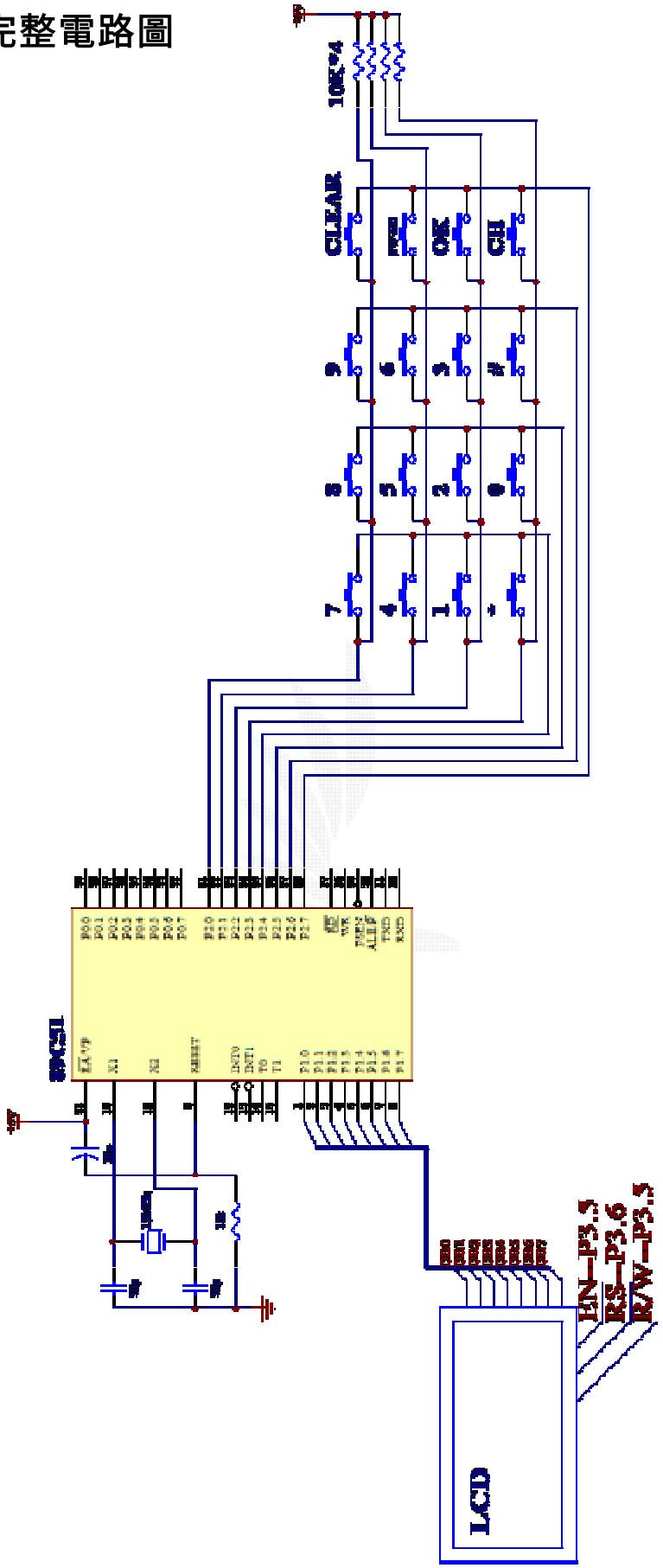


圖 6-2 矩陣鍵盤電路圖

上圖鍵盤功能表如下所示：

0　9 數字鍵	一般輸入密碼的數字鍵。
＊和＃字鍵	加入兩字元的密碼鍵。
CLEAR 鍵	清除＼重新輸入
OK 鍵	確定鍵
CH 鍵	防鎖裝置

圖 6-3 完整電路圖



6-2-2 LCD 接腳定義

這裡以最典型的 20 字X2 列的 LCD 顯示器作為感測系統的顯示裝置，圖(7)即為顯示裝置 LCD 模組，表為其接腳功能表。

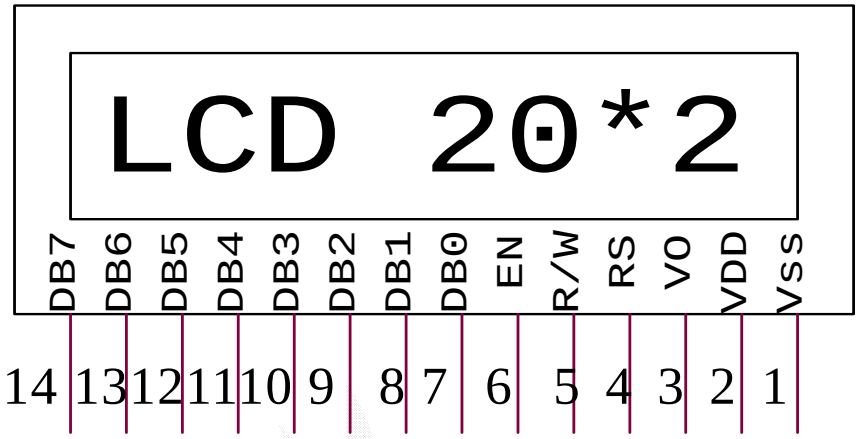


圖 6-4 LCD

表 6-1 20 字X2 列 LCD 接腳功能表

接腳名稱	接腳號碼	接腳功能說明
VSS	1	電源接地腳。
VDD	2	+5V 電源。
Vo	3	亮度電壓調整輸入。
RS	4	暫存器選擇信號：RS=0，選擇指令暫存器。 RS=1，選擇資料暫存器。
R/W	5	讀/寫選擇信號：R/W=0，將資料寫入 LCD。 R/W=1，從 LCD 讀取資料。
E	6	動作致能腳。
DB0 ~ DB7	7 ~ 14	資料匯流排。

6-2-3 LCD 模組與 8051 介面

LCD 與 8051 之介面如圖 6-6 所示，其中包括 RS、E、R/W 等控制腳及資料線的接法。

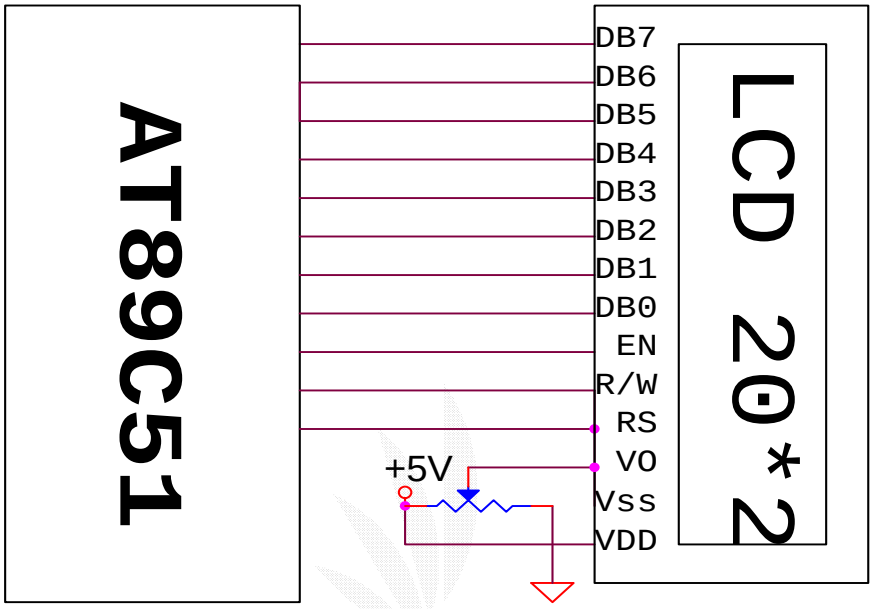


圖 6-5 LCD 與 AT89C51

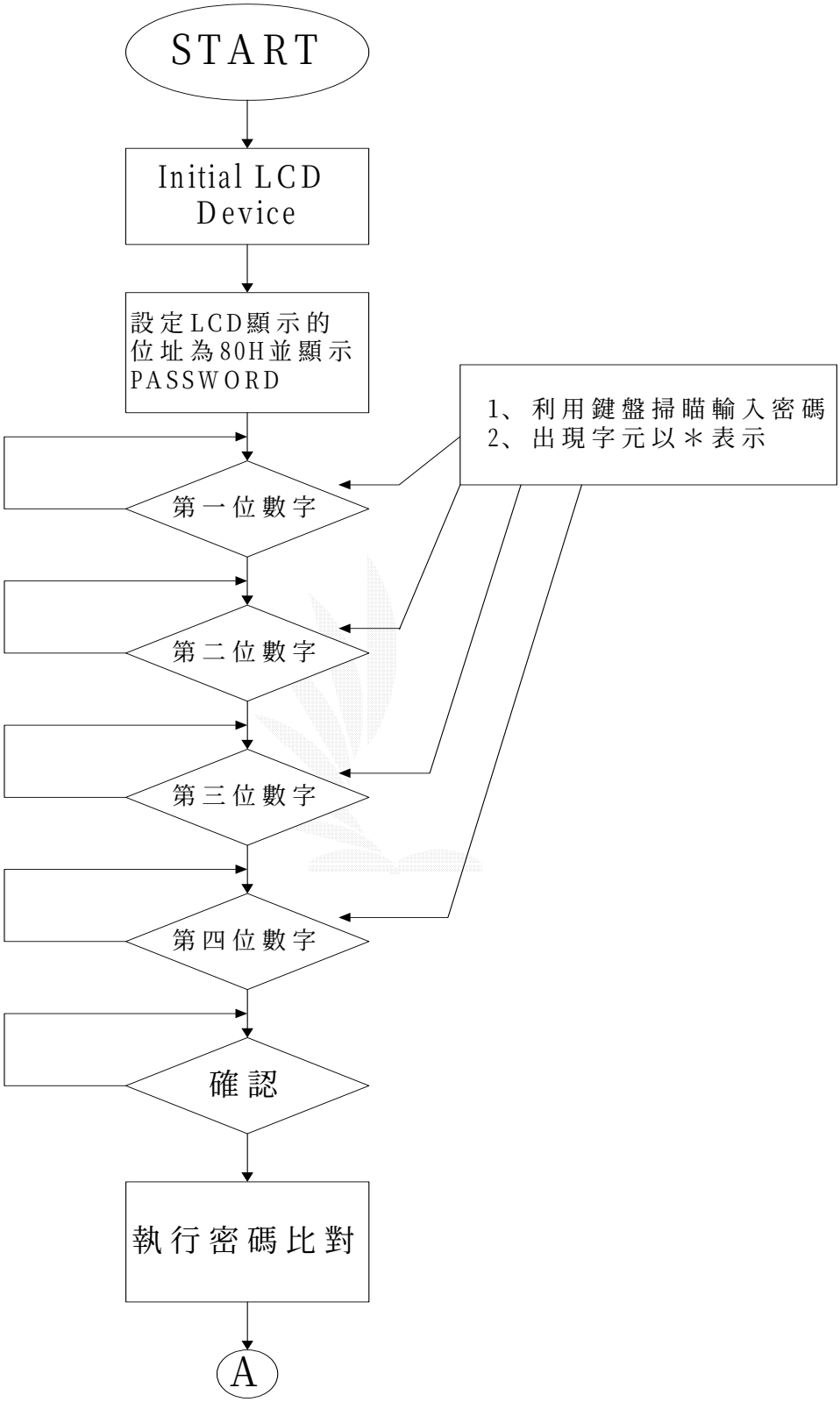
6-2-4 LCD 模組控制

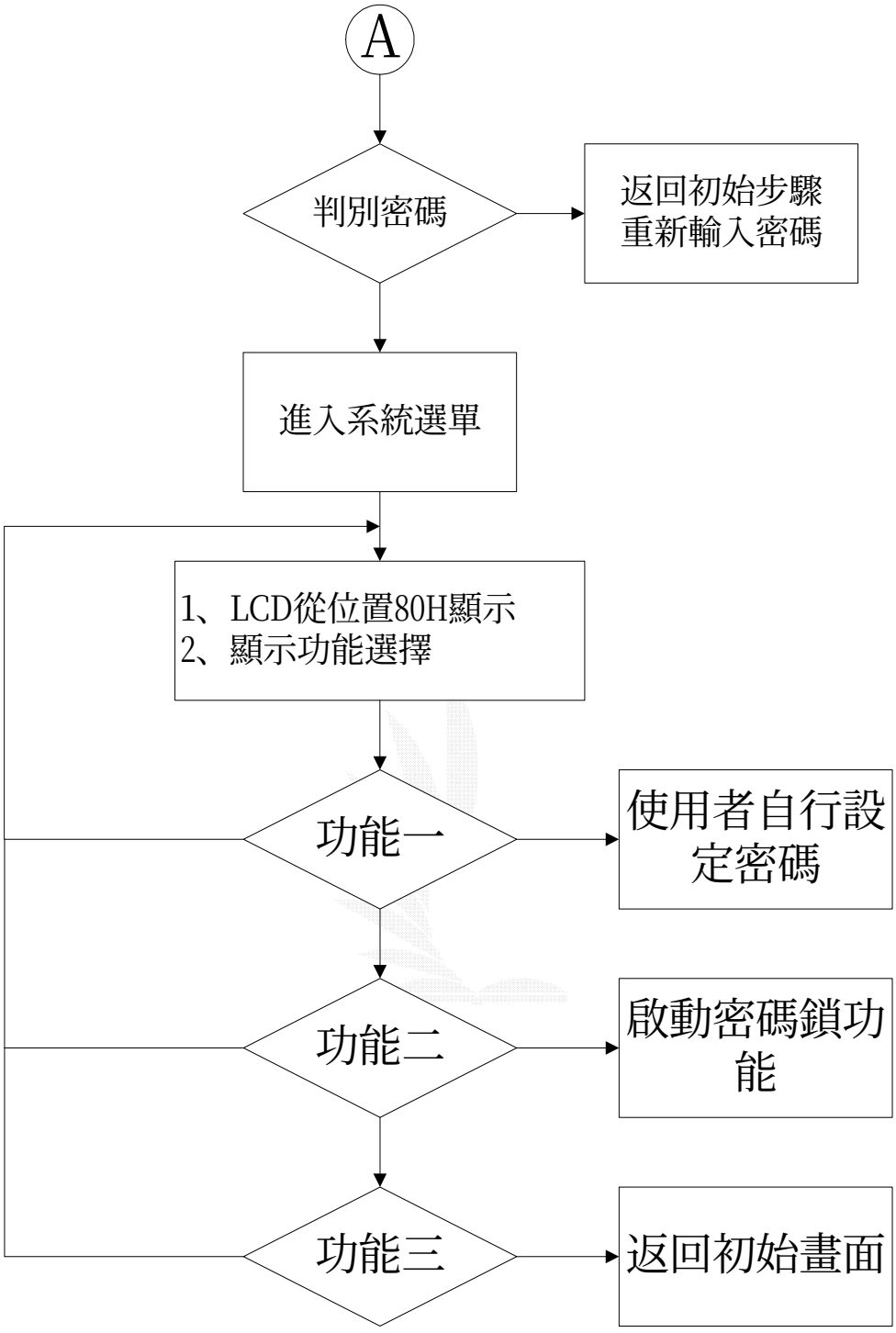
LCD 顯示器主要是利 8051 來控制，寫 51 程式時，先規劃 LCD 上的三隻腳，即 RS、R/W 及 EN。51 程式的接腳規劃如下程式碼所示：

```
RS      BIT     P1.0      ; 將 P1.0 設定為 RS
R_W     BIT     P1.1      ; 將 P1.1 設定為 R_W
EN      BIT     P1.2      ; 將 P1.2 設定為 EN
```

宣告接腳定義，可以使程式設計更容易，也方便日後的維護。

6-3 程式流程圖


START
Initial LCD Device
設定LCD顯示的位址為80H並顯示PASSWORD
1、利用鍵盤掃瞄輸入密碼
2、出現字元以＊表示
第一位數字
第二位數字
第三位數字
第四位數字
確認
執行密碼比對
A


A
判別密碼
返回初始步驟
重新輸入密碼
進入系統選單
1、LCD從位置80H顯示
2、顯示功能選擇
功能一
使用者自行設定密碼
功能二
啟動密碼鎖功能
功能三
返回初始畫面

第七章　心得

完成了此專題後，我才徹底的發現到 8051 這雖是一小小的單晶片，但卻有如此強大的功能及變化，其常使用於一些日常生活的種種物品，包含防盜的電路、馬達的控制、計時器等，透過電路的設計，加上所設計的程式碼燒錄在單晶片上，就可有多種的變化。

專題製作中也遇到了不少的問題，包括了程式碼的正確性，及電路圖的焊接方面，索性之前有組裝過電子套件，加上同學的幫忙查看下才將程式碼等問題一一解決，有了這一次製做專題的經驗後，對於作報告、電路的焊接，及單晶片的功用方面有了更進一步的了解，相信以後如果再碰到相關於單晶片的問題時，必再也不會感到如此的陌生害怕了。

附錄(A)程式碼

```
rs              bit             p3.7
rw              bit             p3.6
enable          bit             p3.5
db0_db7   equ           p1
keyport         equ             p2
num1      equ           21h
num2      equ           22h
num3      equ           23h
num4      equ           24h
addr      equ           25h
words           equ             26h
count     equ           27h

;================================================

                org             00h
                jmp             start
                org             50h
start:
                mov       sp,#60h
                call            initial
                call            cls
                mov       a,#10000000b
                call            write_instruction
                mov       dptr,#line1
```

```
                call            pr_string

loop:
                call            key_scan
                mov     r6,20h
                cjne    r6,#01110111b,m1
                jmp             start
m1:
                cjne    r6,#01111011b,m2
                jmp             loop
m2:
                cjne    r6,#01111101b,m3
                jmp             loop
m3:
                cjne    r6,#01111110b,m4
                jmp             loop
m4:
                mov     num1,20h
                mov     r0,#11001100b           ;r0         is
address
                mov     r1,#00101010b           ;r1         is
word_code
                call            lcddisplay
x1:
                call            key_scan
                mov     r6,20h
            cjne        r6,#01110111b,n1
```

```
                jmp             start

n1:
                cjne    r6,#01111011b,n2
                jmp             x1
n2:
                cjne    r6,#01111101b,n3
                jmp             x1
n3:
                cjne    r6,#01111110b,n4
                jmp             x1
n4:
                mov     num2,20h
                mov     r0,#11001101b           ;r0     is
address
                mov     r1,#00101010b           ;r1     is
word_code
                call            lcddisplay
x2:
                call            key_scan

                mov     r6,20h

                cjne    r6,#01110111b,o1
                jmp             start
o1:
                cjne    r6,#01111011b,o2
```

```
                jmp             x2
o2:
                cjne            r6,#01111101b,o3
                jmp             x2
o3:
                cjne            r6,#01111110b,o4
                jmp             x2
o4:
                mov             num3,20h
                mov             r0,#11001110b       ;r0         is
address
                mov             r1,#00101010b       ;r1         is
word_code
                call            lcddisplay
x3:
            call                key_scan

                mov             r6,20h

                cjne            r6,#01110111b,q1
                jmp             start
q1:
                cjne            r6,#01111011b,q2
                jmp             x3
q2:
                cjne            r6,#01111101b,q3
                jmp             x3
```

```
q3:
                cjne    r6,#01111110b,q4
                jmp             x3
q4:
                mov     num4,20h
                mov     r0,#11001111b           ;r0     is
address
                mov     r1,#00101010b           ;r1     is
word_code
                call            lcddisplay
x4:
                call            key_scan
                mov     r6,20h
                cjne    r6,#01110111b,x5
                jmp             start
x5:
                cjne    r6,#01111101b,x4
                call            check_code      ;check  user
password

                jb              f0,funselect
                ljmp    start

funselect:
                mov     dptr,#line2
                call            string1
```

```
                call        key_scan
                mov     r6,20h
w1:
                cjne    r6,#11101101b,w2
                mov     dptr,#line2
                call        string1
                mov     dptr,#line3
                call        string2

                call        key_scan
                mov     r6,20h
                cjne    r6,#01111101b,funselect
                call        user_code           ;set
user code
                jmp         funselect
w2:
                cjne    r6,#11011101b,w3
                mov     dptr,#line2
                call        string1
                mov     dptr,#line4
                call        string2

                call        key_scan
                mov     r6,20h
                cjne    r6,#01111101b,funselect
                call        start_sys           ;start
system
```

```
                jmp             funselect
w3:
                cjne    r6,#10111101b,funselect
                mov     dptr,#line2
                call            string1
                mov     dptr,#line5
                call            string2

                call            key_scan
                mov     r6,20h
                cjne    r6,#01111101b,funselect
                jmp             start                   ;reset
system

;================================================

lcddisplay:
                mov     a,r0
                call            write_instruction
                mov     a,r1
                call            writelcddata
                ret

string1:
                call            cls
                mov     a,#10000000b
                call            write_instruction
```

```
                call        pr_string
                ret

string2:
                mov     a,#11000000b
                call        write_instruction
                call        pr_string
                ret

cls:
                mov     a,#00000001b
                call        write_instruction
                ret

pr_string:
                clr         a
                movc    a,@a+dptr
                jz          end_pr
                call        writelcddata
                inc         dptr
                jmp         pr_string
end_pr:
                ret

check_code:
                mov     dptr,#password
```

```
        mov     r0,#21h
        mov     r1,#0
        mov     r3,#4
ch1:
        mov     a,@r0
        mov     r2,a

        mov     a,r1
        movc    a,@a+dptr
        xrl         a,r2
        jnz         ch2
        jmp         ch4
ch2:
        mov     dptr,#line6
        call        string1
ch3:
        call        key_scan
        mov     r6,20h
        cjne    r6,#01110111b,ch3
        clr         f0
        jmp         end_check

ch4:
        inc         r0
        inc         r1
        djnz    r3,ch1
        setb    f0
```

```
end_check:
                ret

user_code:
                mov     dptr,#line9
                call            string1
                mov     words,#30h
                mov     count,#0
                mov     addr,#11001111b
u6:
                call            key_scan
                mov     r6,20h

                cjne    r6,#01110111b,u1
                jmp             user_code
u1:
                cjne    r6,#01111011b,u2
                jmp             user_code
u2:
                cjne    r6,#01111101b,u3
                jmp             u5
u3:
                cjne    r6,#01111110b,u4
                jmp             user_code
u4:
                mov     r0,words
                mov     @r0,20h
```

```
                mov     a,addr
                call            write_instruction
                mov     a,#2ah
                call            writelcddata

                inc             words
                inc             count
                dec             addr

                jmp             u6

u5:
                ret

start_sys:
                mov     dptr,#line7
                call            string1
                mov     dptr,#line8
                call            string2
st1:
        call            key_scan
                mov     a,#01110111b        ;if     push
clear botton
                xrl             a,20h
                jnz             st2
                jmp             end_start_sys
```

```
st2:
        mov     a,#01111101b        ;if push ok botton
        xrl     a,20h
        jnz     st1

        mov     dptr,#line11
        call    string1
st3:
        call    key_scan
        mov     a,#11111111b
        xrl     a,20h
        jnz     st4
        jmp     st3
st4:
        call    inputusercode

        call    checkusercode

        jnb     f0,st4

        mov     p0,#00000000b

        mov     dptr,#line12
        call    string1
st5:
        call    key_scan
```

```
                mov     a,#11111111b
                jnz             end_start_sys
                jmp             st5

end_start_sys:
                ret

inputusercode:
                mov     dptr,#line10
                call            string1

                mov     words,#40h
                mov     addr,#11001111b

i6:
                call            key_scan
                mov     r6,20h

                cjne    r6,#01110111b,i1
                jmp             user_code
i1:
                cjne    r6,#01111011b,i2
                jmp             user_code
i2:
                cjne    r6,#01111101b,i3
        jmp             u5
i3:
```

```
                cjne    r6,#01111110b,i4
                jmp             user_code
i4:
                mov     r0,words
                mov     @r0,20h

                mov     a,addr
                call            write_instruction
                mov     a,#2ah
                call            writelcddata

                inc             words
                dec             addr

                jmp             i6
i5:
                ret

checkusercode:
                mov     r0,#30h                 ;usercode
                mov     r1,#40h                 ;inputcode
                mov     r3,count
userch1:
                mov     a,@r1
                mov     r2,a

                mov     a,@r0
```

```
                xrl         a,r2
                jnz         userch2
                jmp         userch4
userch2:
                mov         dptr,#line6
                call        string1
userch3:
                call        key_scan
                mov         r6,20h
                cjne        r6,#01110111b,userch3
                clr         f0
                jmp         end_usercheck

userch4:
                inc         r0
                inc         r1
                djnz        r3,userch1
                setb        f0
end_usercheck:
                ret

initial:
                mov         a,#00111000b
                call        write_instruction

                mov         a,#00001100b
```

```
                call        write_instruction

                mov     a,#00000110b
                call        write_instruction
                ret

checkbusy:
                mov     r0,a
checkbusyloop:
                mov     p3,#01011111b
                setb    enable
                mov     a,p1
                clr         enable
                jb          acc.7,checkbusyloop
                mov     a,r0
                call        delay
                ret

write_instruction:
                call        checkbusy
                mov     p3,#00011111b
                setb    enable
                mov     p1,a
                clr         enable
                ret

writelcddata:
```

```
                call        checkbusy
                mov         p3,#10011111b
                setb        enable
                mov         db0_db7,a
                clr         enable
                ret

delay:
                mov         r6,#5
d1:
            mov         r7,#248
                djnz        r7,$
                djnz        r6,d1
                ret
key_scan:
relax_key:
                mov         keyport,#11110000b

                mov         a,keyport
                xrl         a,#11110000b

                jnz         relax_key

scan_key:
                mov         r1,#11111111b
                clr         c
                mov         r2,#4
```

```
scan_next_column:
                mov     a,r1
                rlc             a

                mov     keyport,a
                mov     r1,a
                mov     a,keyport
                xrl             a,r1

                jz              no_key_pushed

                mov     r5,#2
                call            delay10m
                mov     a,keyport
                xrl             a,r1
                jnz             key_pushed

no_key_pushed:
                djnz    r2,scan_next_column
                jmp             scan_key

key_pushed:
                mov     20h,keyport     ;20h is exist the
key_value
                ret

delay10m:
```

```
                mov     r6,#100
ld1:
                mov     r7,#48
                djnz    r7,$
                djnz    r6,ld1
                djnz    r5,delay10m
        ret

codevalue:
                db
line1:
                db      "PASSWORD",00h
line2:
                db      "FUNCTION SELECT",00H
line3:
                db      "1.CHANGE CODE",00H
line4:
                db      "2.START SYSTEM",00H
line5:
                db      "3.RESET SYSTEM",00H
line6:
                db      "CODE IS ERROR",00H
line7:
                db      "DO YOU WANT TO",00H
line8:
                db      "START SYSTEM ?",00H
line9:
```

```
                db          "SET USER CODE",00H
line10:
                db          "INPUT USER CODE",00H
line11:
                db          "SYS IS STARTING!",00H
line12:
                db          "CODE IS CORRECT!",00H
password:
                db
        11101101b,11101101b,11011101b,10111011b
                end
```